Database Processing
CS 451 / 551

Lecture 3:

Storage

Suyash Gupta

Assistant Professor

Distopia Labs and ORNG

Dept. of Computer Science

(E) suyash@uoregon.edu

(W) gupta-suyash.github.io

UNIVERSITY OF OREGON

Assignment 1

- **Assignment 1 will be released today on course webpage:**
  - **https://gupta-suyash.github.io/cs451-fall25.html**

- **Deadline: October 28, 2025 at 11:59pm.**

- Start planning, talk to your group members.
  - **If you don't have a group yet, today is your last day to form/join a group.**
  - **We will no longer create groups for you.**

- We will discuss Assignment 1 in the next class.

- Feel free to read **Lstore paper** → We will discuss in the class.

Last Class

- SQL Querying

Physical View

- Until now we studied the logical or a user’s view of the data.

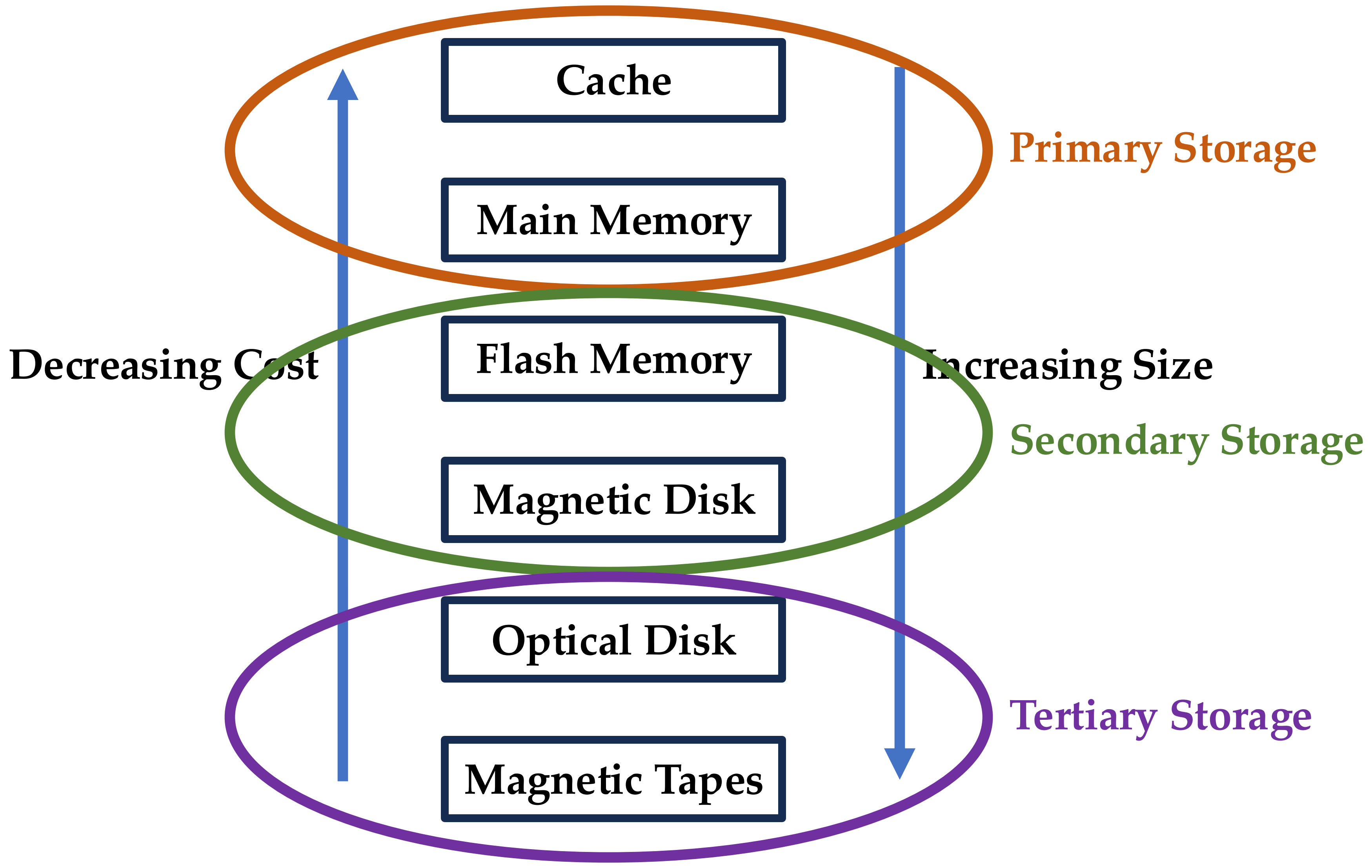
- Next, we look at the physical view of the data?

Physical View

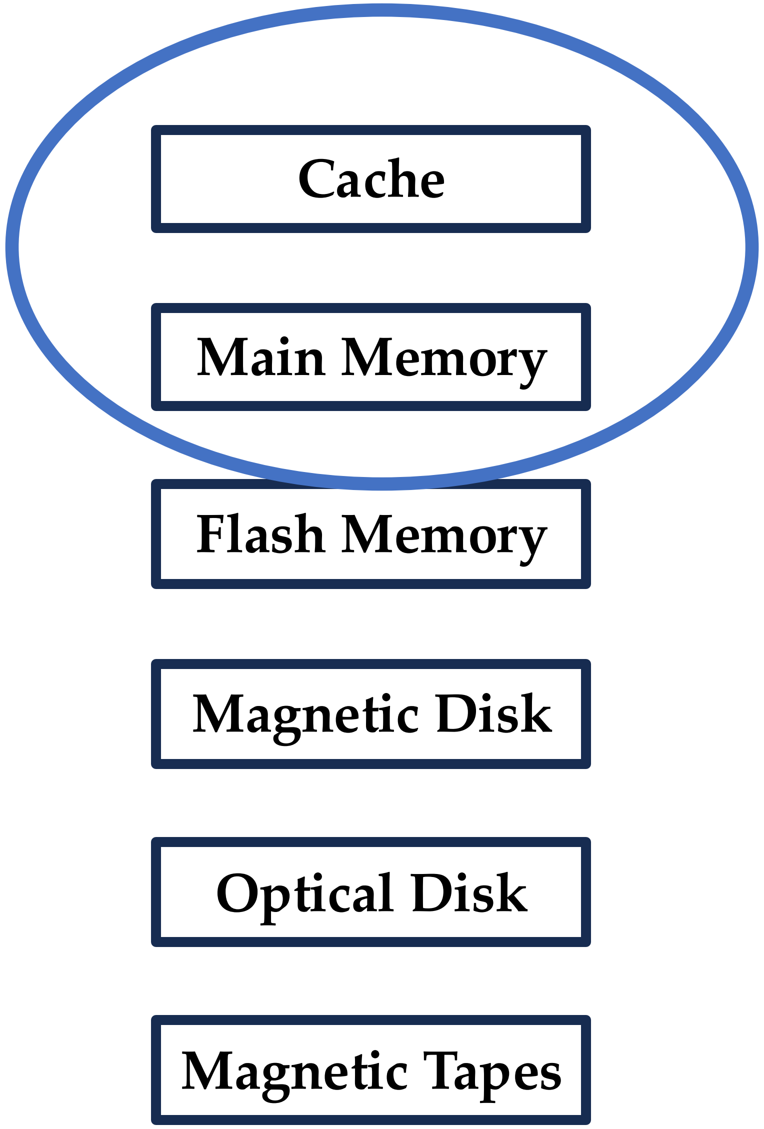
- Until now we studied the logical or a user's view of the data.

- Next, we look at the physical view of the data.
  - How is data stored?
  - What are the physical media to store data?
  - What data structures help in faster access?

Storage Hierarchy



Storage Hierarchy



Always inside your computer.

Too small to store the full database.

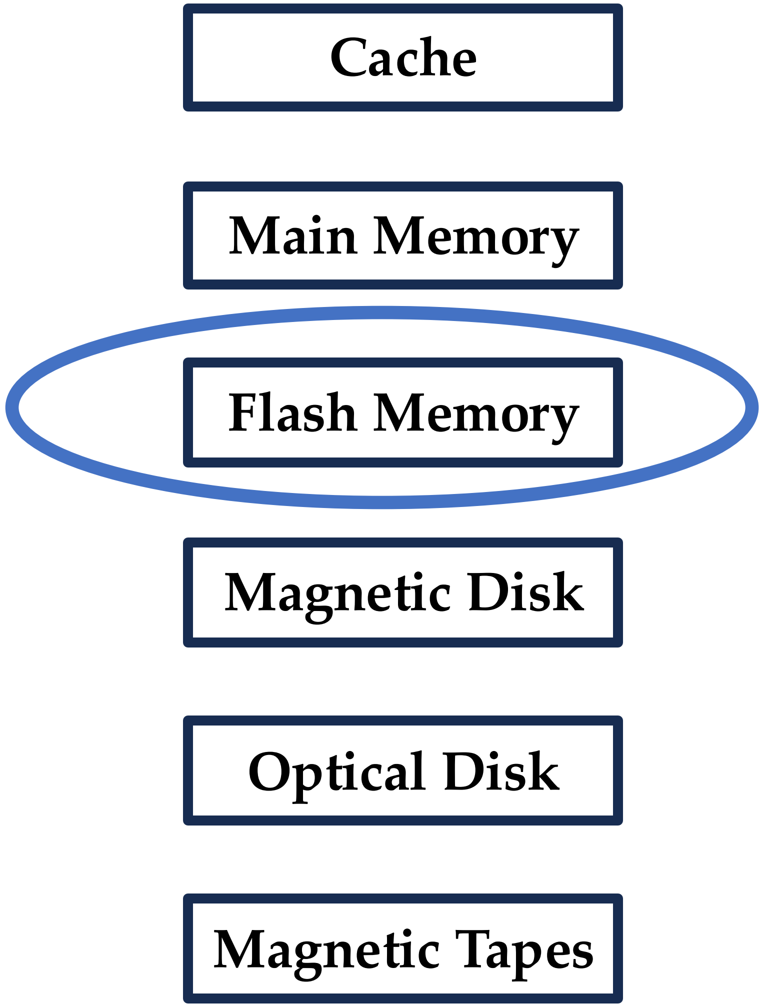
Data loss when power turns off.

Still valuable as the CPU aims to frequently loads them up with recent data to avoid data misses.

A good application or system design targets cache hits.

Often, there are several levels of main memory: L1, L2, and L3.

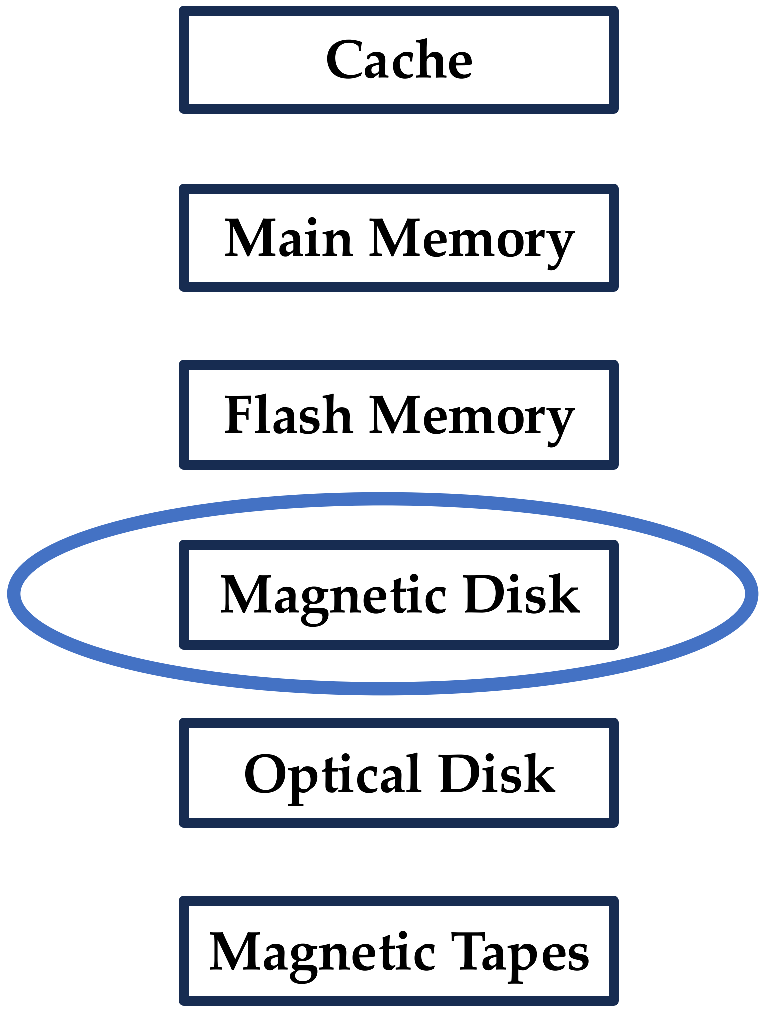
Storage Hierarchy



USB stick or external memory.

No data loss on turning off power.

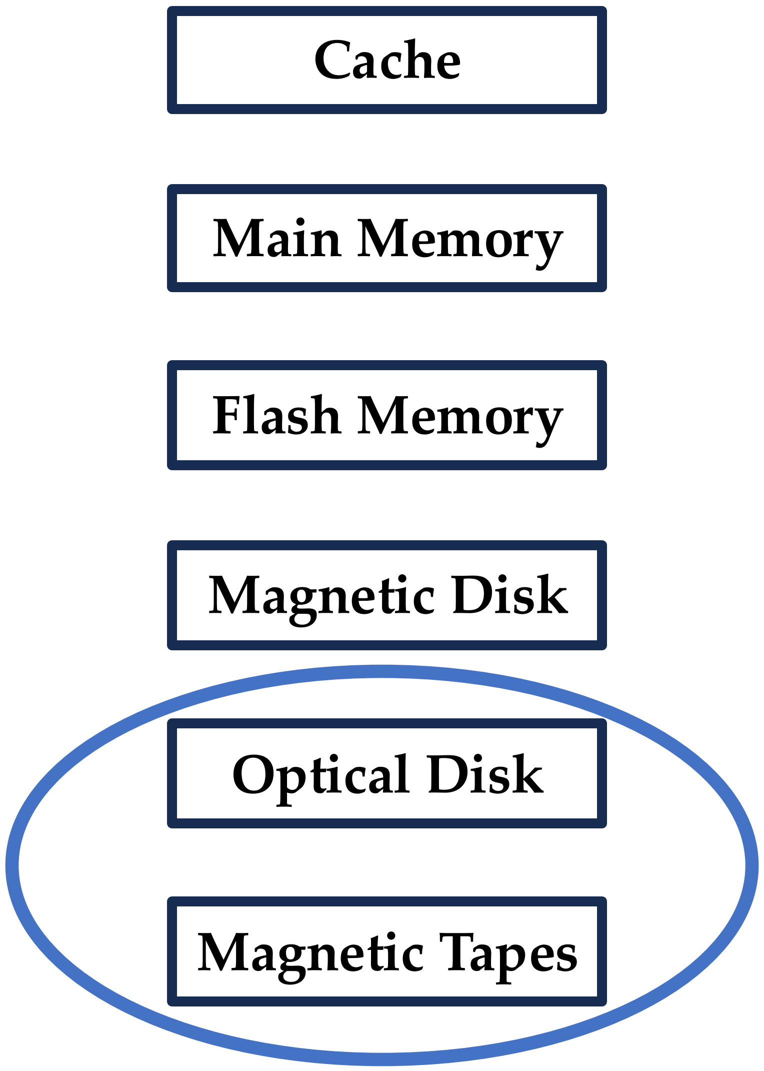
Storage Hierarchy



Useful for long-term storage of the database.

At the time of processing, CPU moves data from magnetic disk to the main memory.

Storage Hierarchy



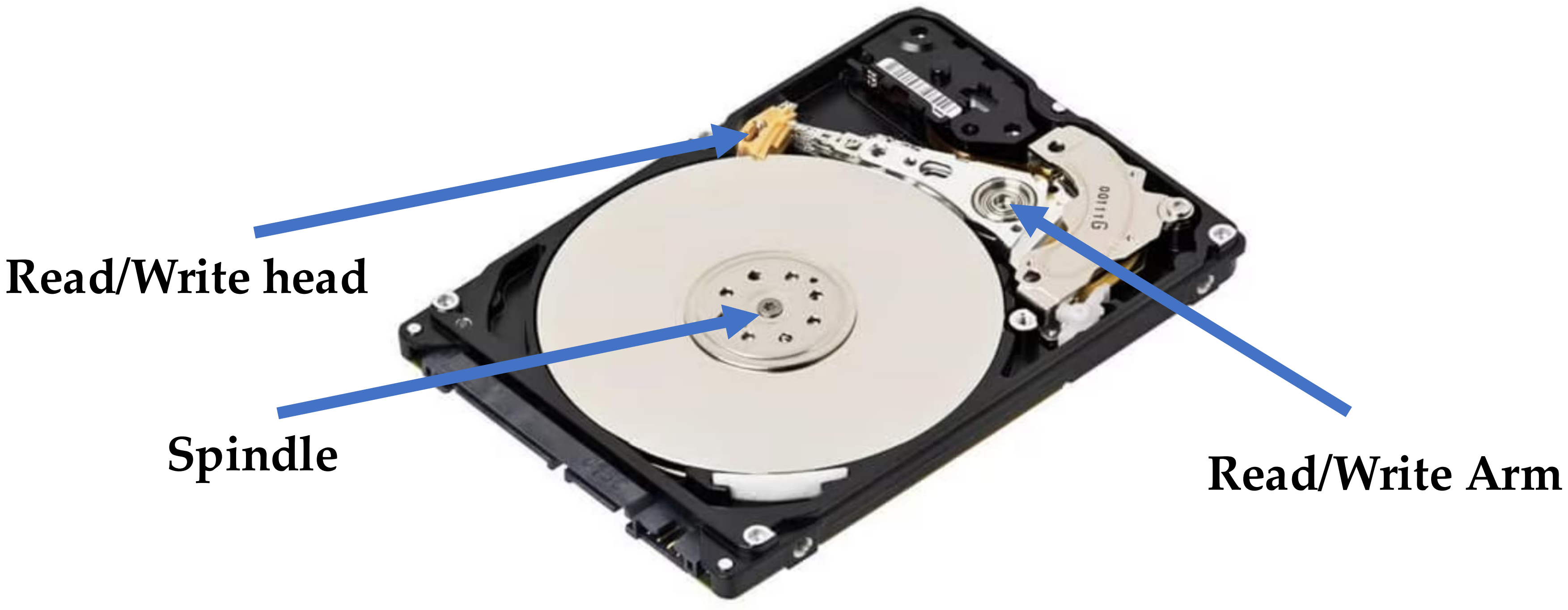
Older storage media; not common everywhere.

Magnetic Disks

- Magnetic disks are where large databases reside; databases can span multiple disks.
- Components of Magnetic Disks?

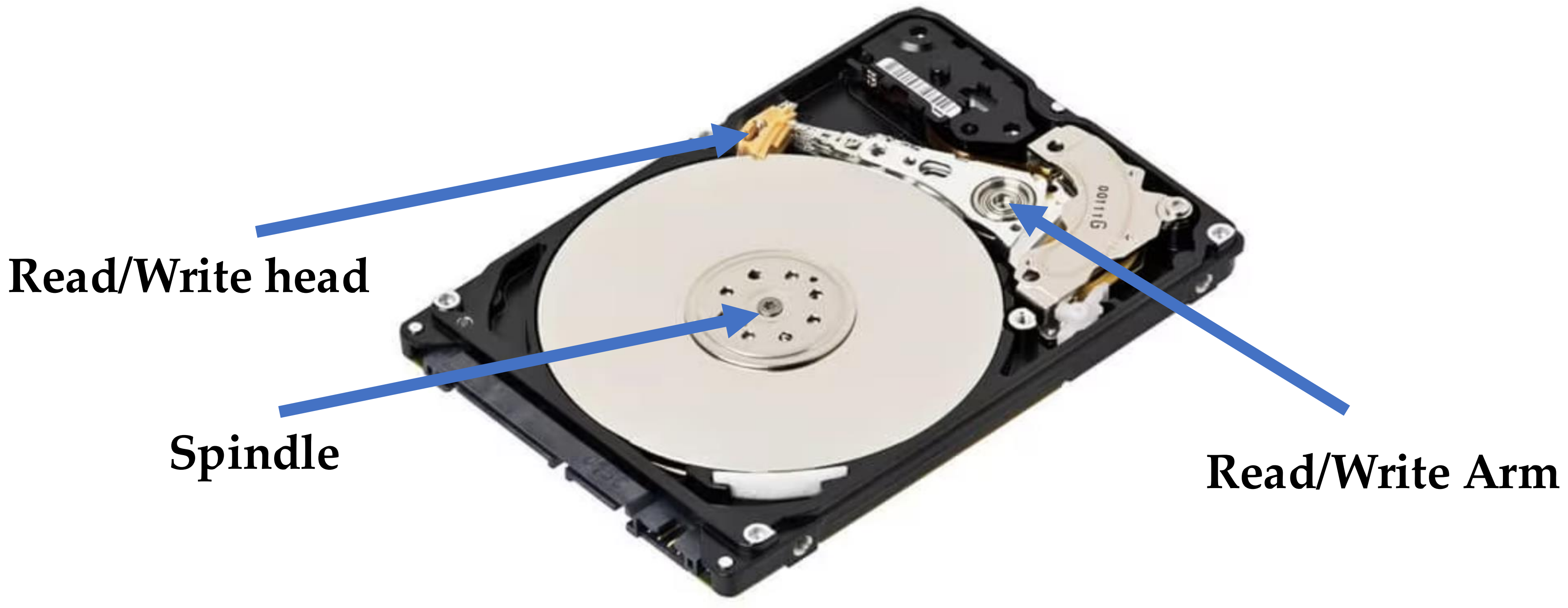
Magnetic Disks

- Magnetic disks are where large databases reside; databases can span multiple disks.
- Components of Magnetic Disks?



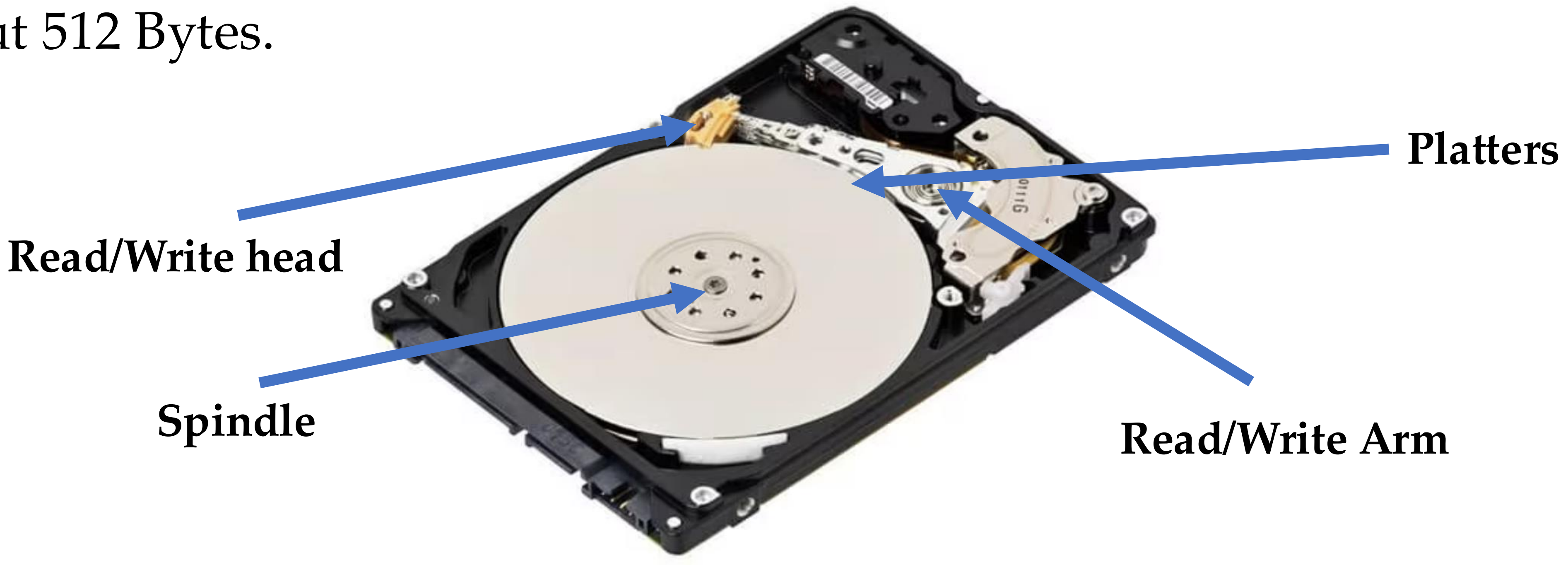
Magnetic Disks

- Magnetic disks are where large databases reside; databases can span multiple disks.
- **How is data stored in Magnetic Disks?**



Magnetic Disks

- Magnetic disks are where large databases reside; databases can span multiple disks.
- Each disk is divided into **platters** → each platter has multiple **tracks** → each track has multiple **sectors**.
  - Platters sit on a **spindle**, which rotates at some speed (say 1500 rpm).
- Sector is the smallest unit of information that can be read or written.
  - About 512 Bytes.



Magnetic Disks – Read-write Head?

Magnetic Disks – Read-write Head?

- Read-write head stores the information on a sector magnetically as reversals of the direction of magnetization.

- Read-write head does not touch the disk, but is kept as close as possible to increase writing density.

Magnetic Disks – Disk Controller?

- Disk controller is a software (stored on disk case) that interfaces between actual disk h/w and the computer system.
  - It accepts high-level commands to read/write to a sector.
  - Attaches a **checksum** to each sector when the data is written.
  - On reading a sector, computes its checksum, and matches against stored checksum.
  - If checksums don't match → **Data corruption!**

Disk Performance Metrics?

Disk Performance Metrics

- **Access Time** – Time difference between when a read/write request is issued to when data transfer begins.
  - **Seek Time + Rotational Latency**
  - Seek Time → Time to position the read-write arm to correct track.
  - Rotational latency → Time to reach the correct sector under the read-write head.

Disk Performance Metrics

- **Access Time** – Time difference between when a read/write request is issued to when data transfer begins.
  - **Seek Time + Rotational Latency**
  - Seek Time → Time to position the read-write arm to correct track.
  - Rotational latency → Time to reach the correct sector under the read-write head.

- **Data Transfer rate** → The rate at which data can be retrieved from a disk.
  - Measured after the first sector is accessed.

Disk Performance Metrics

- **Access Time** – Time difference between when a read/write request is issued to when data transfer begins.
  - **Seek Time + Rotational Latency**
  - Seek Time → Time to position the read-write arm to correct track.
  - Rotational latency → Time to reach the correct sector under the read-write head.

- **Data Transfer rate** → The rate at which data can be retrieved from a disk.
  - Measured after the first sector is accessed.

- **Mean Time To Failure (MTTF)** → The average amount of time a device runs continuously without any failure.

RAID

RAID

- Disks are slow to read/write the data!
  - How can we speed up?

- **Redundant Arrays of Independent Disks (RAID)**
  - Multiple disks instead of one.

  - Can read or write to the disks in **parallel**.

  - Can improve **reliability** by duplicating same data to multiple disks
    → Also known as replication
    → Guards against failures!
    → Increases Mean Time to Failure

How do Modern Databases Look Like?

How do Modern Databases Look Like?

- There are databases which are quite small → up to 50 GB!
  - Some of these you can even store in **main-memory**!
  - For example: 2009 English Wikipedia was just 20GB!

- Then there are other databases which can be stored in **Flash drives (SSDs).**
  - Slower than main-memory but faster than HDDs (magnetic disks)

- But, several other databases require petabytes of space**.**
  - Facebook image database!
  - You would need **multiple disks**!

Disk vs. OS Terminology

- From disk perspective, we store data on platters → tracks → sectors.

- From OS perspective, we store data in **blocks or pages**.

- One block can span multiple sectors!
  - In this class, we will assume each block has same size as a sector.

- Blocks and pages are **synonyms**!
  - **More technically →** Blocks is used as the terminology for data on disk and page is used when data is in main memory (RAM).

What is meant by next block?

What is meant by next block?

- You want to read some blocks from disk. How should you proceed?

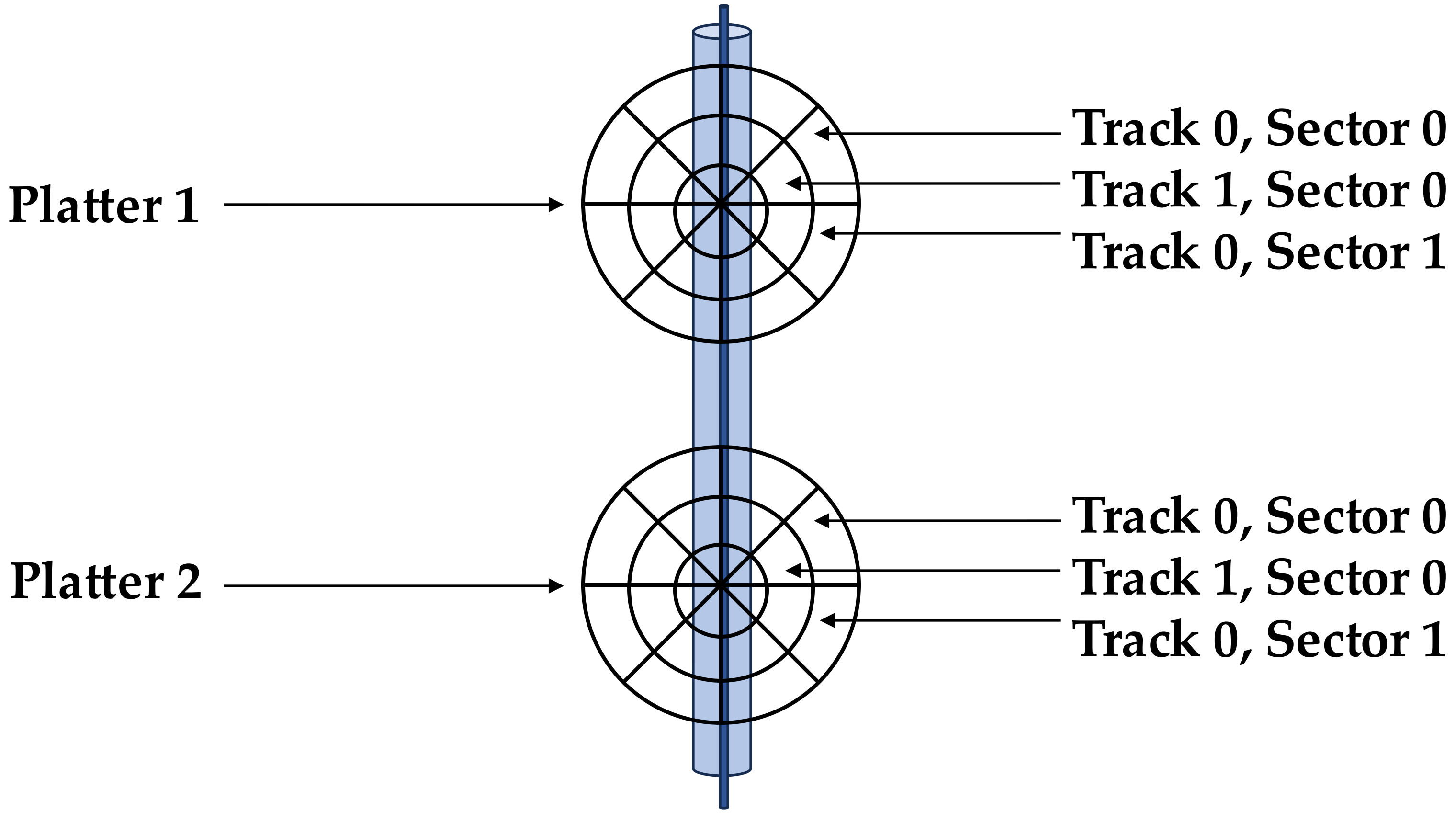
What is meant by next block?

- You want to read some blocks from disk. How should you proceed?
  - Find the first block address you are interested.
  - Move the read/write arm to specific platter and then to correct track.
  - Then move the read/write head to correct sector.

- Now, where do you read the next block? Which sector?
  - You want to read or write fast! Not increase seek time + rotational latency!

What is meant by next block?

- Lets define next block to read
  - Read sectors on same track.
  - Then same track number on next platter (same cylinder, a logical concept)
  - Then next track on adjacent cylinder.



What is meant by next block?

- This is how we even store database files on disk!

- Files are stored **sequentially** on disk.
    - Allows us to fetch using the notion of next block.

- Any other benefits of sequential storage?

What is meant by next block?

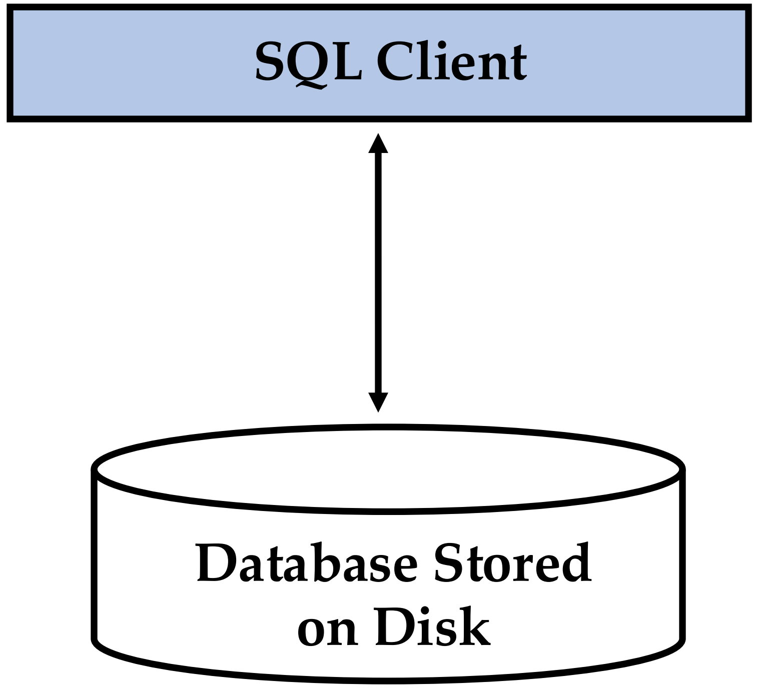
- This is how we even store database files on disk!

- Files are stored **sequentially** on disk.
  - Allows us to fetch using the notion of next block.

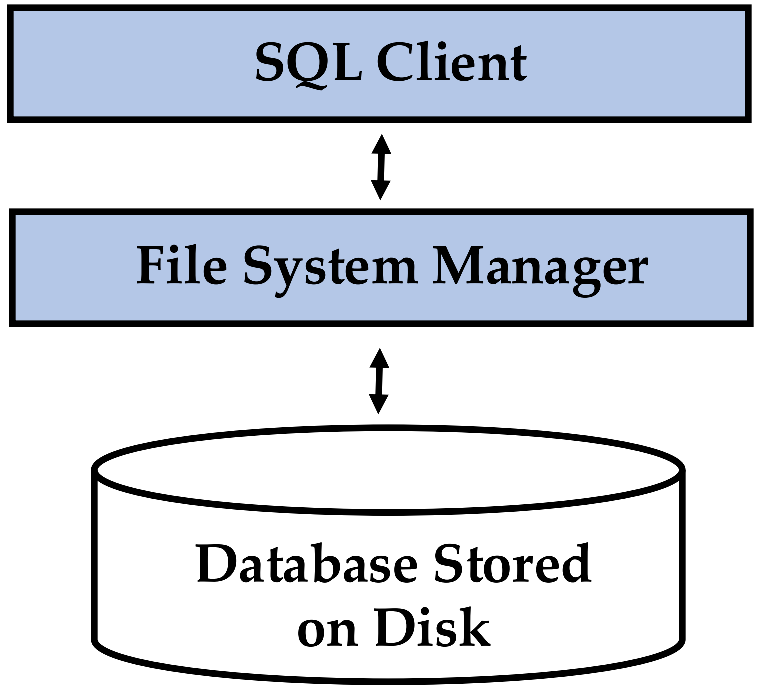
- Any other benefits of sequential storage?

- Sequential scan also allows **pre-fetching**!
  - Read a large amount of data ahead of time even if we need only one block.
  - Hope that **future reads will be cheap** → data available in memory.

Disk and System Communication



Disk and System Communication

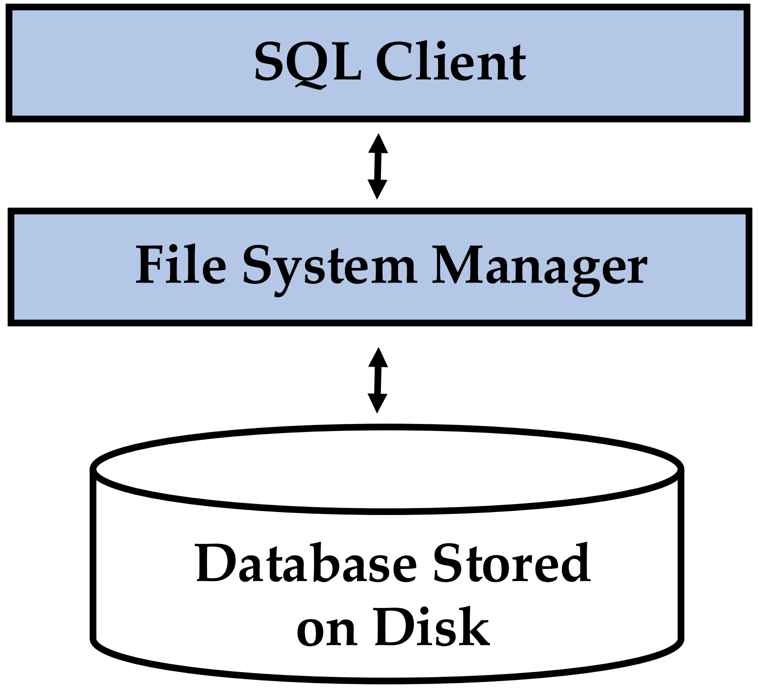


Disk and System Communication

- File system manager, a component of OS manages interaction with disk.
  - Keeps track which page is where on disk.
  - Read a page from the disk.
  - Write a page to the disk.

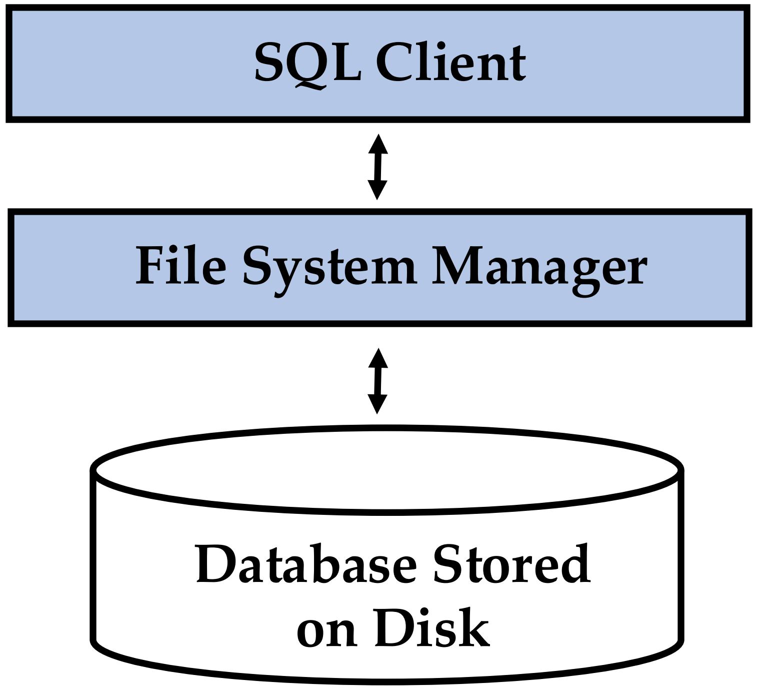
- Higher levels basically send requests to the file system manager.
  - They ask filesystem to allocate or deallocate logical pages.
  - Read or write data to pages.

- Remember higher levels do not know anything about how pages are managed or how the database is stored!

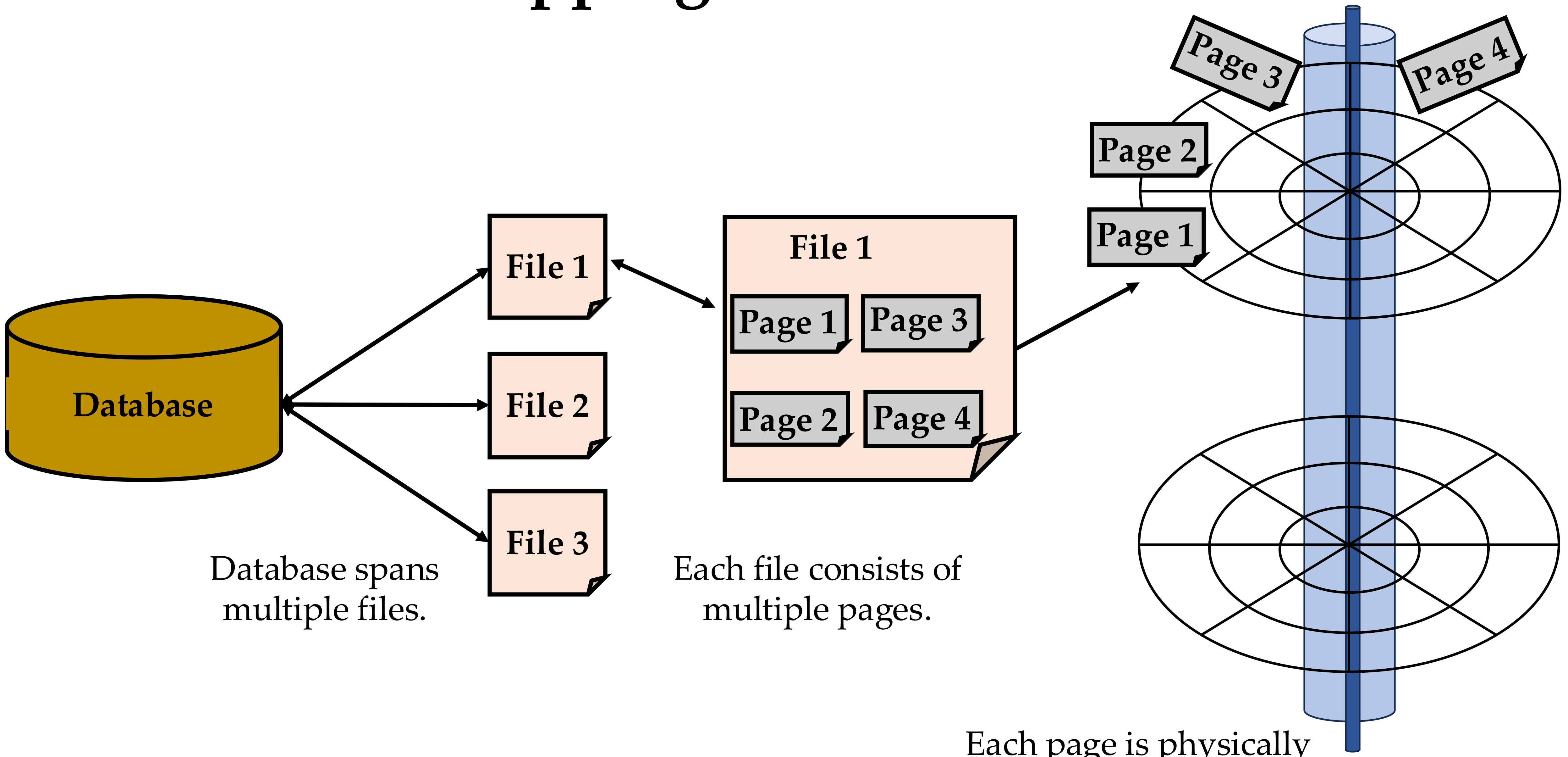


Disk and System Communication

- Thus, File system manager needs to provide access to pages as fast as possible!
- It needs to ensure that pages are stored sequentially on this disk.
- It should be able to pre-fetch pages to fulfill requirements fast!
- Any pre-fetched pages are pushed to the memory so that future reads can be served from the memory itself.



Mapping Tables to Disk



Database spans multiple files.

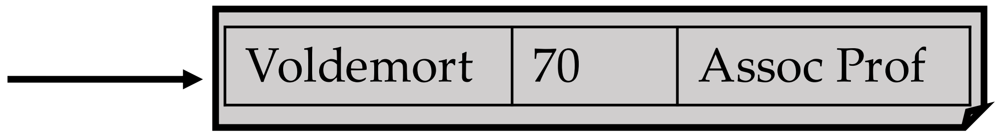
Each file consists of multiple pages.

Each page is physically stored on a disk sector.

Mapping Tables to Disk

name	age	title
Voldemort	70	Assoc Prof
Anakin	20	Assoc Prof
Kang	20	Prof
Gru	45	Asst Prof
Thanos	100	Prof

cs-employees



Each record is stored in a page. We are assuming one page per record.

Each table consists of several records/rows.

Mapping Records to File

- So how do we place records on a file?

Mapping Records to File

- So how do we place records on a file?
  - We need to determine **the size of the record** to know how many blocks it needs.
  - Not every record is exactly one block size!

- Records can be divided into two groups:
  - Fixed-length records
    - Record Size can be determined from the schema.

  - Variable-length records
    - Size of some attributes can only be determined at runtime.

Fixed-Length Records

```
create table cs_employees
        (       name   char(23),
                age    int,
                salary numeric(8,2)
        );
```

record 0	**Voldemort**	**70**	**400**
record 1	Anakin	20	200
record 2	Kang	20	500
record 3	Gru	45	100
record 4	Thanos	100	400

cs-employees

- Say, the following is the schema of our table.
- **Total size** of a record = 23 Bytes + 4 Bytes + 8 Bytes = 35 Bytes.
- How do we store in the memory?
    - Reserve first 35 Bytes for Record 1, next 35 Bytes for Record 2, and so on.

Problems with Fixed-Length Records

- Unless block size is a multiple of 35 Bytes, some records will need to be stored in another block.
    - So two blocks need to be fetched for accessing each record.

Problems with Fixed-Length Records

- Deleting a record creates a problem of space management!

Problems with Fixed-Length Records

- Deleting a record creates a problem of space management!

- **Way 1:**
  - On deleting the record, move all the succeeding records up.
  - Shift one space in the memory.
  - Simple solution!
  - **Bad performance**.
  - Large number of records moved.

record 0	**Voldemort**	**70**	**400**
record 1	Anakin	20	200
record 3	Gru	45	100
record 4	Thanos	100	400

cs-employees

On deleting record 2 and moving 3 and 4 up.

Possible Solution for Fixed-Length Records

- **Way 2:** Insert a new record in the space of deleted record.
  - Hard to track, which all spaces are available.

record 0	**Voldemort**	**70**	**400**
record 1	Anakin	20	200
record 5	Joker	66	700
record 3	Gru	45	100
record 4	Thanos	100	400

cs-employees

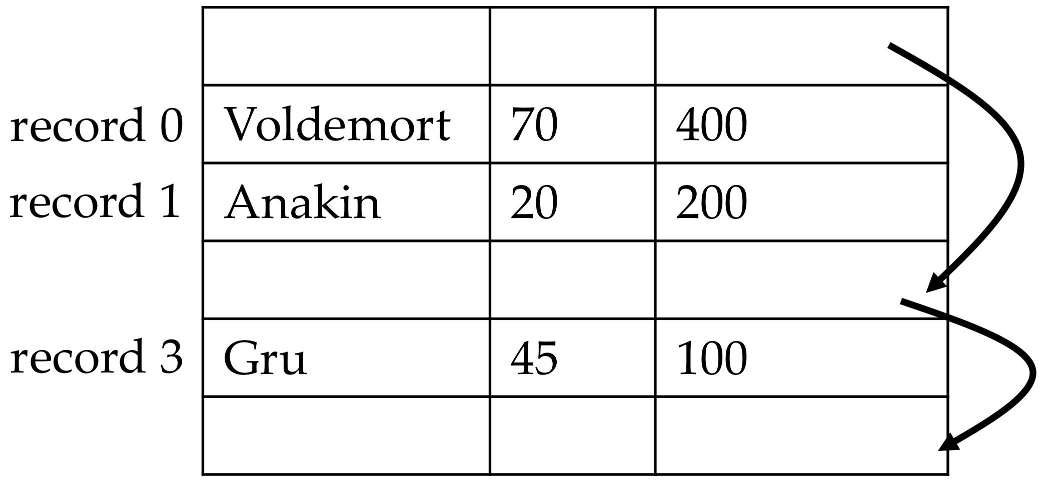
New record 5, inserted in space of record 2.

Free List for Fixed-Length Records

Free List for Fixed-Length Records

- Make use of **file header**.
  - File header stores all the meta (important) information about the file.

- We can maintain a **free-list**: store here information about first deleted record.
  - Then this record points to next deleted record.



record 0	Voldemort	70	400
record 1	Anakin	20	200
record 3	Gru	45	100

cs-employees

Free-list, tracked through header.

Variable-Length Records

```
create table cs_employees
        (       name   varchar(100),
                age    int,
                salary numeric(8,2)
        );
```

record 0	**Voldemort**	**70**	**400**
record 1	Anakin	20	200
record 2	Kang	20	500
record 3	Gru	45	100
record 4	Thanos	100	400

cs-employees

- Say, the following is the schema of our table.
- The maximum size for varchar is 100 Bytes, but a name could be smaller.
- We can say that each record has size (100 Bytes + 4 Bytes + 8 Bytes) = 112 Bytes.
- But record 0 has size only 18 Bytes? Why waste so much space?

Representing Variable-Length Records

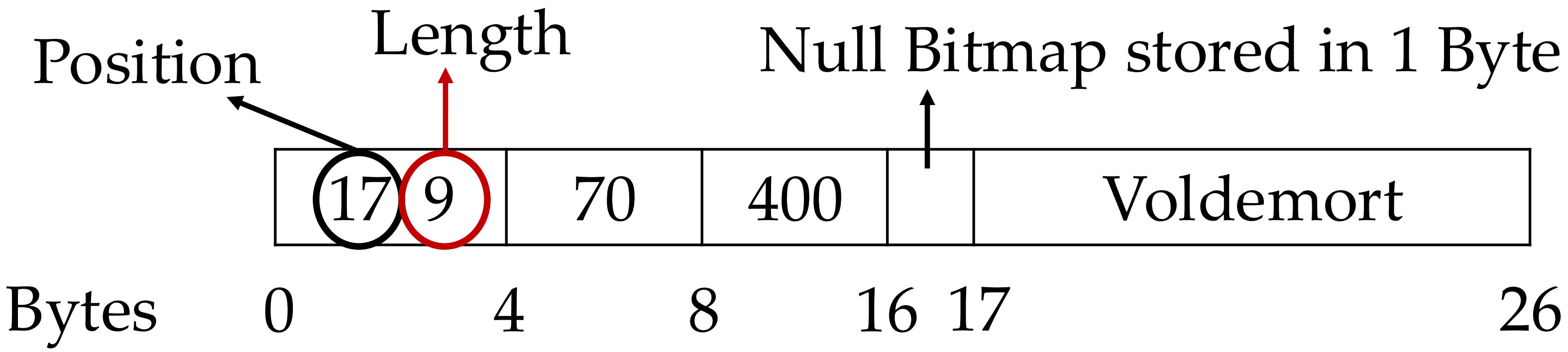
Representing Variable-Length Records

- For each variable-length record, store an **(offset, length)** pair.
  - Offset → The position in the representation.
  - Length → The size of variable-length field.

Representing a Record

```
create table cs_employees
(        name   varchar(100),
         age    int,
         salary numeric(8,2)
);
```

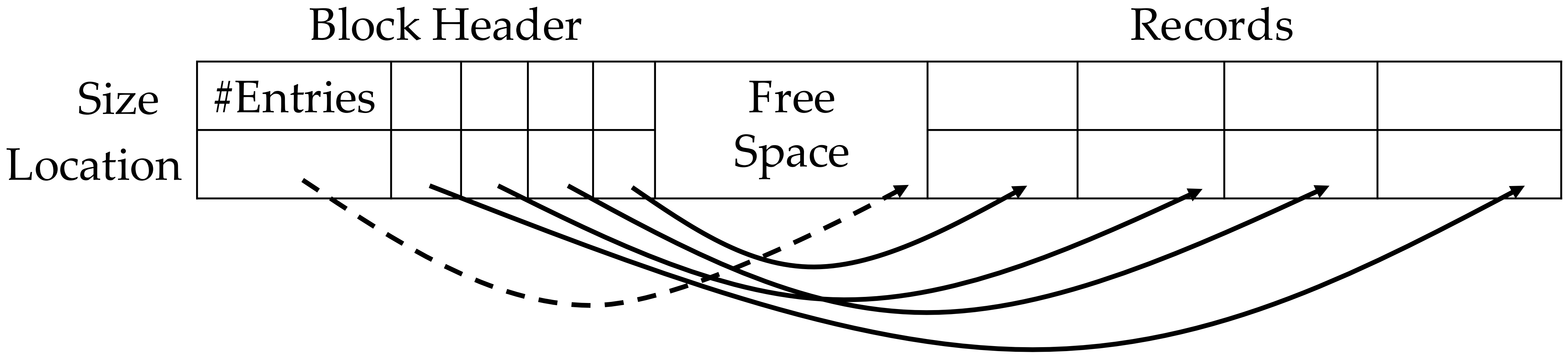
Voldemort	**70**	**400**
Anakin	20	200
Kang	20	500
Gru	45	100
Thanos	100	400



Mapping Records to Disk Blocks

- We assumed that one record per disk block.
  - Often, multiple records per block.
  - How to store or find a record in a block?

- Each block has a header that tracks **number of records and free space**.



Mapping Records to File

- So we have mapped records to a block/page.
- But we still have not mapped them to files.
- Different ways of organizing records to a file?

Mapping Records to File

- So we have mapped records to a block/page.

- But we still have not mapped them to files.

- Different ways of organizing records to a file:
    - Heap File organization
    - Sequential File organization
    - Hashing File organization

Heap File Organization

- Any record can be placed anywhere in the file.
- No ordering of records.
- One file per relation.

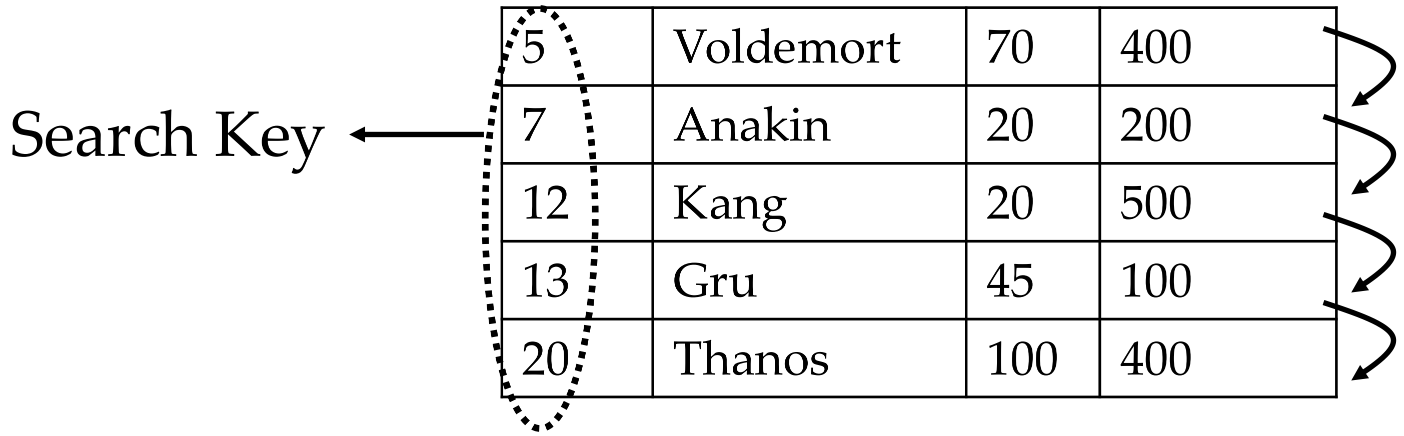
Sequential File Organization

Sequential File Organization

- Allowing quickly searching records.

- Records are stored in sorted order based on some search key.
  - Search key does not need to be a primary key.

- Records are linked via pointers for access and stored physically in search-key order.



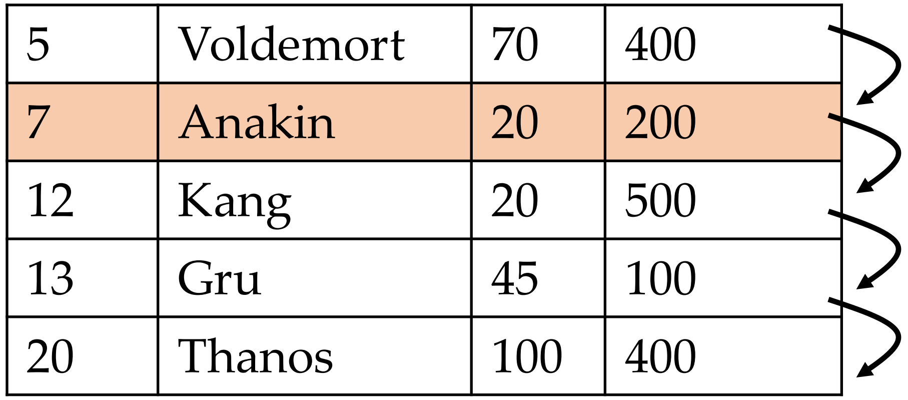
5	Voldemort	70	400
7	Anakin	20	200
12	Kang	20	500
13	Gru	45	100
20	Thanos	100	400

Sequential File Organization

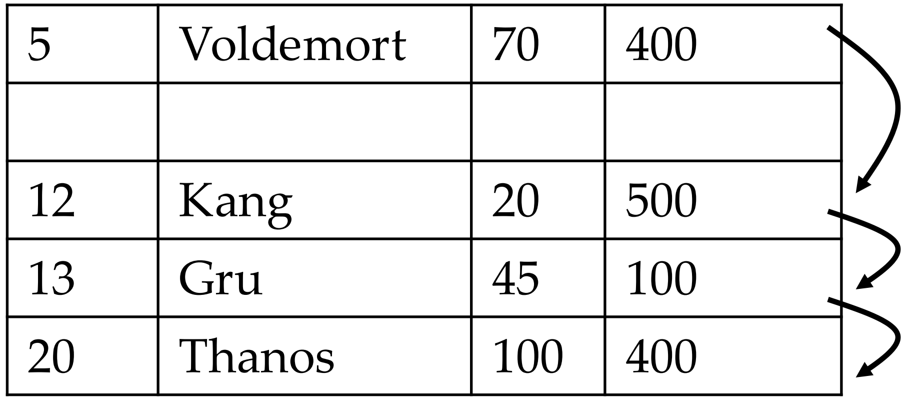
- Sorted Order creates challenges?

Sequential File Organization

- Sorted Order creates challenges.
- **Deleting a record** leaves a space and pointer needs to be remapped.
- Say we want to delete record 7.



5	Voldemort	70	400
7	Anakin	20	200
12	Kang	20	500
13	Gru	45	100
20	Thanos	100	400



5	Voldemort	70	400
12	Kang	20	500
13	Gru	45	100
20	Thanos	100	400

Sequential File Organization

- How about insertion?

- Say we want to insert a record 6.

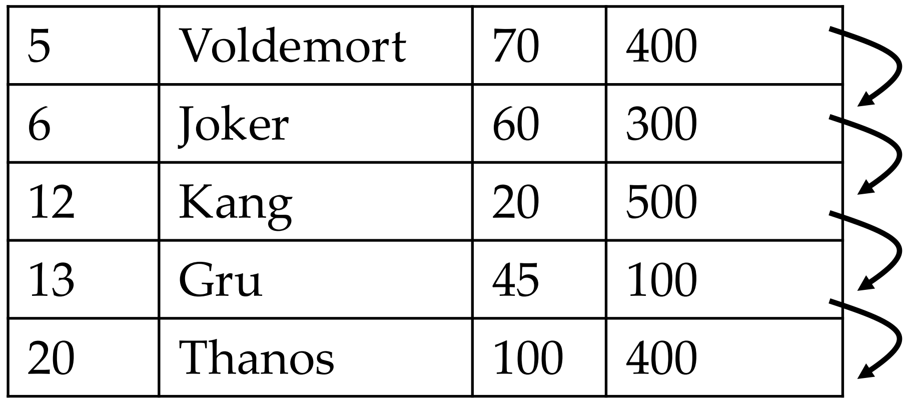
- **This case is easy as we have an open gap in the file**.

5	Voldemort	70	400
12	Kang	20	500
13	Gru	45	100
20	Thanos	100	400

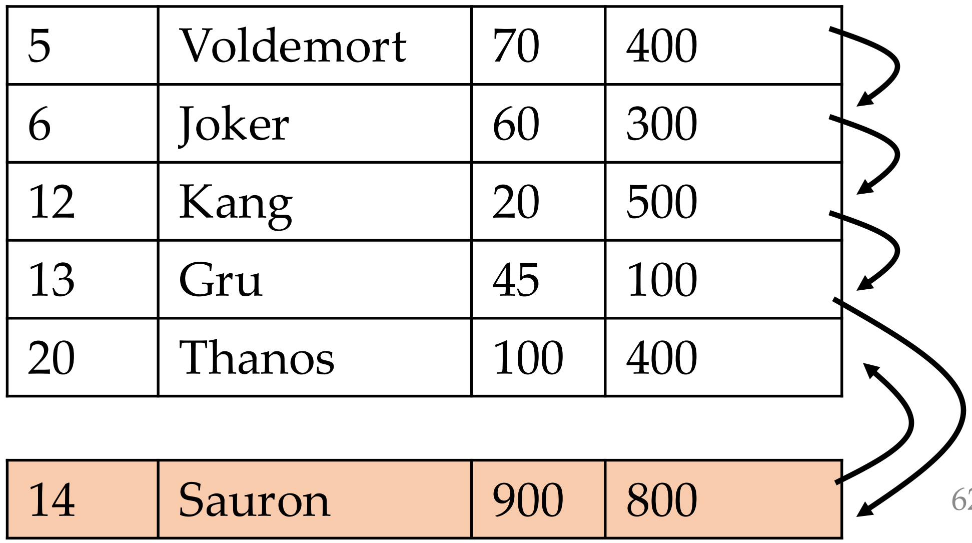
5	Voldemort	70	400
6	Joker	60	300
12	Kang	20	500
13	Gru	45	100
20	Thanos	100	400

Sequential File Organization

- How about inserting a record when there is no open gap?
- Say we want to insert a record 14.
- **We need to insert the new record in an overflow block and re-map pointers.**



5	Voldemort	70	400
6	Joker	60	300
12	Kang	20	500
13	Gru	45	100
20	Thanos	100	400



5	Voldemort	70	400
6	Joker	60	300
12	Kang	20	500
13	Gru	45	100
20	Thanos	100	400

14	Sauron	900	800

Sequential File Organization

- If the overflow block ends up having too many records → File Reorganization
- Too expensive to reorganize complete file.

Database Buffers

Database Buffers

- DBMS aims to minimize the access between disk and memory.

- If your database is larger than your main memory (generally the case), it needs to be stored on disk.

- Disks are slow → Main memory is fast!

- So we have a **buffer manager**, which manages what blocks should be present in a part of memory termed as **buffer**.

- Buffer manager is like virtual memory manager in OS.
    - Tries to pre-fetch blocks → predicts what blocks will be needed in future to save access time.

Database Buffers - Working

Database Buffers - Working

- When database asks for a block from buffer manager:
  - If block already present in the buffer → It forwards the pointer to the block.
  - If block is not present in the buffer
    - It **discards some existing block** to make space.
    - Fetches the required block from disk.
    - Forwards the pointer to the block.

- **How to discard a block**.
  - If the block in the buffer has been updated, that is, the version on disk is stale, then replace the version on disk with the updated version.
  - If the block in the buffer has no new updates, then simply discard.

Database Buffers

- Which block to discard → **Buffer Replacement Policies**
  - **Least Recently Used (LRU)**
    - Same as the OS LRU policy.
    - The block least recently accessed is discarded first.
  - **Toss Immediate Policy**
    - If you can determine a block will not be used again, discard it.

select * from cs_employees **natural join** department;

Here, once a tuple of cs_employees is accessed, it can be discarded